Performance evaluation

We're going to evaluate portfolio performance. The common way to do this is to estimate a portfolio's return adjusted for risk using a factor model with tradeable risk factors.

What's a risk factor? These risk factors are portfolios and the idea is that the expected excess return on these risk factors is compensation to investors for bearing the risk inherent in holding those portfolios. For the return variation in these factors to be "risky", it should be something that investors cannot easily diversify. If it was easy to diversify, then investors could put a small bit of the asset in their portfolio and capture the return without affecting portfolio volatility. That would imply being able to increase return without adding risk. Hence the requirement that a factor constitute return variation that is hard to diversify away.

The greater the riskiness of the factor, the greater the factor's expected return (i.e., the risk-return tradeoff). For example, most people feel that stocks are riskier than bonds and indeed stocks have historically had higher returns than bonds.

The risk adjustment involves estimating a portfolio's β's on different risk factors. These β's constitute the exposure of the portfolio to the risk factor. If the factor return goes up by 1%, then the portfolio's return goes up by $\beta \times 1\%$.

We can estimate these β's by OLS regressions of the portfolio's returns on contemporaneous returns of the risk factors. The slope coefficients on the risk factors are the portfolio's betas on the risk factors. The regression intercept is known as α. It represents the average return of the portfolio that is not explained by the portfolio's β's on the risk factors. This alpha is the risk-adjusted return.

Intuitively, α is the average return on a portfolio long the investment you are evaluating and short a portfolio with the same factor risk as that portfolio. If the factors and factor betas accurately measure the portfolio's risk, then the alpha is the portfolio's return that is unrelated to the portfolio's risk. Investors like positive alphas because that implies that the portfolio's return is higher than what investors require for bearing the portfolio's risk.

One thing to keep in mind is that throughout this discussion, we have discussed things from the perspective of arbitrage. That is, like a trader. We have not made any assumptions about utility functions or return distributions. This is the Arbitrage Pricing Theory (APT) of Stephen Ross (1976). He was motivated by the observation that

> "... on theoretical grounds it is difficult to justify either the assumption [in mean-variance anlysis and CAPM] of normality in returns...or of quadratic preferences...and on empirical grounds the conclusions as well as the assumptions of the theory have also come under attack."

The APT way of thinking is less restrictive than economically motivated equilibrium asset pricing models. Which is nice. But it has the cost that it does not tell us as much. With the APT we cannot say precisely what a security's return should be. We can only say that if we go long a portfolio and short the portfolio that replicates its factor exposure, then the alpha shouldn't be *too* big. But if we're thinking like a trader, that's perhaps most of what we care about anyway.

```
#r "nuget: FSharp.Stats, 0.4.1"
#r "nuget: FSharp.Data"

#load "../common.fsx"

open System
open FSharp.Data
open Common

open FSharp.Stats

Environment.CurrentDirectory <- __SOURCE_DIRECTORY__
```

We get the Fama-French 3-Factor asset pricing model data.

```
let ff3 = French.getFF3 Frequency.Monthly
```

Let's get our factor data.

```
let myFactorPorts = CsvProvider<"myExcessReturnPortfolios.csv",
                                ResolutionFolder = __SOURCE_DIRECTORY__>.GetSample()
```

ML.NET is a .NET (C#/F#/VB.NET) machine learning library. There are several tutorials and many F# examples in the sample github repository here.

We will use ML.NET for Ordinary Least Squares (OLS) regression, but you can also do pretty fancy machine learning models with it. So think of it as a gentle introduction giving you some guidance on how to use ML.NET. This will help if you want to experiment with fancy machine learning models after you're done with this course.

```
#r "nuget:Microsoft.ML,1.5"
#r "nuget:Microsoft.ML.MKL.Components,1.5"

open Microsoft.ML
open Microsoft.ML.Data
```

Let's start with our long-short portfolio.

```
let long = myFactorPorts.Rows |> Seq.filter(fun row -> row.PortfolioName = "Mine" && row.Index = Some 3)
let short = myFactorPorts.Rows |> Seq.filter(fun row -> row.PortfolioName = "Mine" && row.Index = Some 1)

type Return = { YearMonth : DateTime; Return : float }
let longShort =
    // this is joining long to short by YearMonth:DateTime
    let shortMap = short |> Seq.map(fun row -> row.YearMonth, row) |> Map
    long
    |> Seq.map(fun longObs ->
        match Map.tryFind longObs.YearMonth shortMap with
        | None -> failwith "probably your date variables are not aligned"
        | Some shortObs -> { YearMonth = longObs.YearMonth; Return = longObs.Ret - shortObs.Ret })
    |> Seq.toArray
```

For regression, it is helpful to have the portfolio return data merged into our factor model data.

```
type RegData =
    // The ML.NET OLS trainer requires 32bit "single" floats
    { Date : DateTime
      Portfolio : single
      MktRf : single
      Hml : single
      Smb : single }

// ff3 indexed by month
// We're not doing date arithmetic, so I'll just
// use DateTime on the 1st of the month to represent a month
let ff3ByMonth =
    ff3
    |> Array.map(fun x -> DateTime(x.Date.Year, x.Date.Month,1), x)
    |> Map

let longShortRegData =
    longShort
    |> Array.map(fun port ->
        let monthToFind = DateTime(port.YearMonth.Year,port.YearMonth.Month,1)
        match Map.tryFind monthToFind ff3ByMonth with
        | None -> failwith "probably you messed up your days of months"
        | Some ff3 ->
            { Date = monthToFind
              Portfolio = single port.Return // single converts to 32bit
              MktRf = single ff3.MktRf
              Hml = single ff3.Hml
              Smb = single ff3.Smb })
```

We need to define a ML.Net "context"

> Once instantiated by the user, it provides a way to create components for data preparation, feature enginering, training, prediction, model evaluation.

```
let ctx = new MLContext()
```

Now we can use the context to transform the data into ML.NET's format.

In the below code, a .NET Enumerable collection is equivalent to an F# sequence. This line says load an F# collection were the elements of the collection are `RegData` records. The part between `<>` is how we define the type of the data on the collection.

```
let longShortMlData = ctx.Data.LoadFromEnumerable<RegData>(longShortRegData)
```

Now we are going to define our machine learning trainer. OLS!

The OLS trainer is documented here with an example in C#. Though C# is not the easiest language to follow.

But these F# regression examples with fancier ML models is easier. So we do the OLS trainer like they do those trainers.

```
let trainer = ctx.Regression.Trainers.Ols()
```

Now we define the models that we want to estimate. Think of this like an ML pipeline that chains data prep and model estimation.

- `Label` is the variable that we are trying to predict or explain with our model.

- `Features` are the variables that we are using to predict the label column.

```
let capmModel = 
    EstimatorChain()
        .Append(ctx.Transforms.CopyColumns("Label","Portfolio"))
        .Append(ctx.Transforms.Concatenate("Features",[|"MktRf"|]))
        .Append(trainer)

let ff3Model = 
    EstimatorChain()
        .Append(ctx.Transforms.CopyColumns("Label","Portfolio"))
        .Append(ctx.Transforms.Concatenate("Features",[|"MktRf";"Hml";"Smb"|]))
        .Append(trainer)
```

Now we can estimate our models.

```
let capmEstimate = longShortMlData |> capmModel.Fit
let ff3Estimate = longShortMlData |> ff3Model.Fit
```

The results can be found in OLSModelParameters Class.

CAPM results.

```
capmEstimate.LastTransformer.Model
```

Fama-French 3-Factor model results

```
ff3Estimate.LastTransformer.Model
```

You will probably see that the CAPM R^2 is lower than the Fama-French R^2. This means that you can explain more of the portfolio's returns with the Fama-French model. Or in trader terms, you can hedge the portfolio better with the multi-factor model.

We also want predicted values so that we can get regression residuals for calculating the information ratio. ML.NET calls the predicted value the score.

The ML.NET OLS example shows getting predicted values using C# with the `context.Data.CreateEnumarable`. Searching the ML.NET samples github repo for `CreateEnumerable` shows F# examples.

```
[<CLIMutable>]
type Prediction = { Label : single; Score : single}

let makePredictions (estimate:TransformerChain<_>) data =
    ctx.Data.CreateEnumerable<Prediction>(estimate.Transform(data),reuseRowObject=false)
    |> Seq.toArray

let residuals (xs: Prediction array) = xs |> Array.map(fun x -> x.Label - x.Score)

let capmPredictions = makePredictions capmEstimate longShortMlData
let ff3Predictions = makePredictions ff3Estimate longShortMlData

capmPredictions |> Array.take 3
```

```
capmPredictions |> residuals |> Array.take 3
```

```
val it : single [] = [|-0.01339226775f; -0.03164776042f; 0.01224330813f|]
```

```
let capmResiduals = residuals capmPredictions
let ff3Residuals = residuals ff3Predictions
```

In general I would write a function to do this. Function makes it a bit simpler to follow. It's hard for me to read the next few lines and understand what everything is. Too much going on.

```
let capmAlpha = (single 12.0) * capmEstimate.LastTransformer.Model.Bias
let capmStDevResiduals = sqrt(single 12) * (Seq.stDev capmResiduals)
let capmInformationRatio = capmAlpha / capmStDevResiduals
```

```
val capmAlpha : single = 0.05657358468f
val capmStDevResiduals : single = 0.09837608039f
val capmInformationRatio : single = 0.5750746131f
```

```
let ff3Alpha = (single 12.0) * ff3Estimate.LastTransformer.Model.Bias
let ff3StDevResiduals = sqrt(single 12) * (Seq.stDev ff3Residuals)
let ff3InformationRatio = ff3Alpha / ff3StDevResiduals
```

```
val ff3Alpha : single = 0.05347375572f
val ff3StDevResiduals : single = 0.08723817766f
val ff3InformationRatio : single = 0.6129627824f
```

```
// Function version

let informationRatio monthlyAlpha (monthlyResiduals: single array) =
    let annualAlpha = single 12.0 * monthlyAlpha
    let annualStDev = sqrt(single 12.0) * (Seq.stDev monthlyResiduals)
    annualAlpha / annualStDev

informationRatio capmEstimate.LastTransformer.Model.Bias capmResiduals
```

```
val informationRatio :
  monthlyAlpha:single -> monthlyResiduals:single array -> single
val it : single = 0.5750746131f
```

```
informationRatio ff3Estimate.LastTransformer.Model.Bias ff3Residuals
```

```
val it : single = 0.6129627824f
```